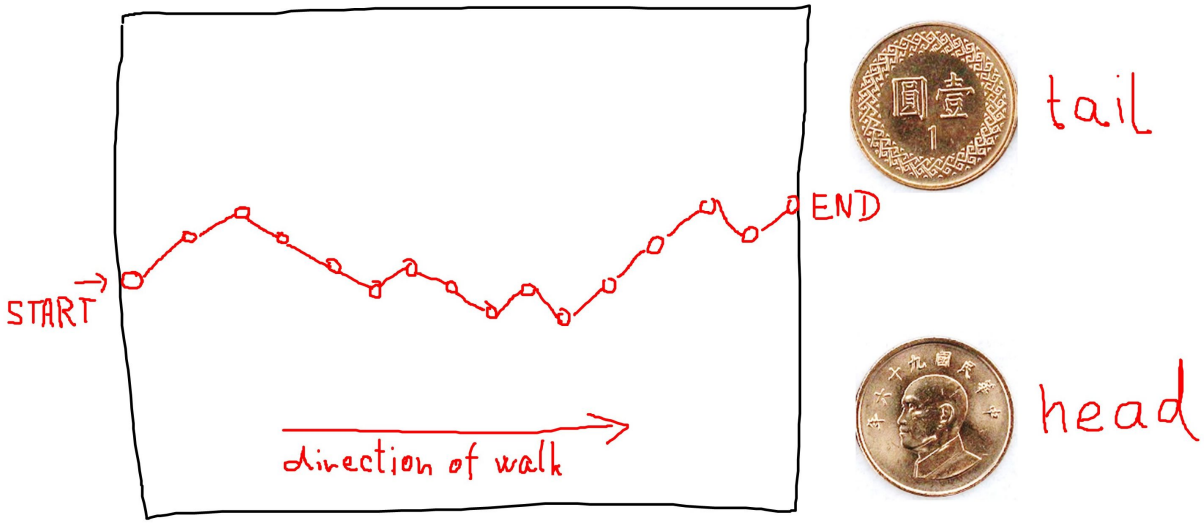
Random walk

The problem

Use R to simulate a random walk. Imagine that you stand at the beginning of the large field, with a coin in your hand. You cannot do a step straight forward - you can only do a step slightly left or slightly right. To decide, you flip the coin - head means right, tail means left. You flip the coin before each new step. If I draw your walk, it will look a little bit if you were drunk:



Please, translate this *random walk* into the R code. First, make a loop which will repeat the 1000 steps forward, each with flipping the coin, and return a vector with positions which can then be plotted. Then, wrap the loop into a function called `random.walk`, which will have the only argument `no.steps` for the number of steps. The function should simulate given number of steps, and return a vector with positions, which can then be plotted.

How to R code it

First, the short description of the problem, which should be translated into a script. Imagine that at the beginning, you are standing at the point zero ($x_1 = 0$). In each subsequent step (x_2, x_3, ...), you go either left (+1) or right (-1), with the choice done random (tossing the coin).

Numerically it will look as following: Generate a vector of the length `no.steps`, containing only zeros. The current (starting) position is zero ($x_1 = 0$). Randomly generate either +1 or -1, add this generated value to x_1 and fill the result at the second field of the vector. Repeat `no.steps - 1` times. Wrap the whole loop into the function `random.walk` and make sure that it has one argument (`no.steps`) and returns a vector with coordinates of the walk which can be plotted.

One would intuitively expect that, because we walk right and left randomly, in total we still head more or less forward. But is it like this?

Creating the script

This description tries to be didactic - I don't go directly to creating the loop, but instead show you how it would look like if you don't use a loop at all. Then, I introduce the `for` loop, and finally wrap everything into the function `random.walk` with the only argument `no.steps`. You may skip directly to the end of it if you are not interested or you already know everything.

Step by step

First, define an empty vector, which will be used to deposit information about your current position:

```
steps.rw <- vector (mode = 'numeric', length = 1000)
```

The first argument indicates that the vector will contain numbers (alternatives would be `character` or `logical`), the second argument defines the length of the vector, i.e. how many elements it will contain.

And now, you start to walk forward. To choose randomly the direction, use function `sample`, with two arguments, x - the vector from which the elements will be randomly selected, and `size` - number of selected elements from x. The following script:

```
sample (x = c(-1,1), size = 1)

[1] -1
```

will randomly select one value from the vector, containing numbers `-1` and 1, in this case value `-1`. You just made the first step to the right.

Vector `steps.rw` contains only zeroes:

```
steps.rw
   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [79] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  ...
  ...
```

First element is therefore zero, which accord to your initial position. The second element can be calculated as following:

```
steps.rw [2] <- steps.rw [1] + sample (c(-1,1), 1)
```

You may have realized that in function `sample`, there are no names of arguments (x and `size`) - they are not needed, until you write the values for arguments in the defined order (what is this order you can check in the help to the function `sample`, which you call typing `?sample`).

Further steps can be defined in the following way:

```
steps.rw [3] <- steps.rw [2] + sample (c(-1,1), 1)
steps.rw [4] <- steps.rw [3] + sample (c(-1,1), 1)
steps.rw [5] <- steps.rw [4] + sample (c(-1,1), 1)
steps.rw [6] <- steps.rw [5] + sample (c(-1,1), 1)
steps.rw [7] <- steps.rw [6] + sample (c(-1,1), 1)
```

```
steps.rw [8] <- steps.rw [7] + sample (c(-1,1), 1)
steps.rw [9] <- steps.rw [8] + sample (c(-1,1), 1)
steps.rw [10] <- steps.rw [9] + sample (c(-1,1), 1)
```

The use of for loop

As you may guess, the approach above wouldn't take you too far, as you would have to repeat the line 1000 times. It's time to introduce function to create loop. The name of the function is `for`:

```
for (k in seq (2, 1000))
  {
  steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
  }
```

The script says the following: the variable k will subsequently be of value 2 to 1000, and with each of this value make the following: at the k-th position of vector `steps.rw` save the value, calculated from k-1 position of vector `steps.rw` by adding randomly selected value `-1` or 1.

```
steps.rw

   [1]   0  -1   0  -1   0  -1  -2  -3  -4  -3  -2  -3  -4  -3  -4  -5  -4  -5  -4
-5  -4  -5  -6  -7  -8  -9 -10  -9 -10 -11 -10  -9 -10 -11 -10  -9 -10 -11 -10
  [40] -11 -10 -11 -10  -9 -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3
4   3   2   1   2   3   2   1   2   3   4   5   6   5   4   3   2   3   2   3
  [79]   4   5   6   7   8   7   8   7   6   7   8   7   8   7   8   7   8   7   6
5   6   5   4   5   6   7   8   9  10   9   8   7   8   9  10  11  10   9   8
 [118]   9   8   9  10  11  10  11  12  11  12  13  12  13  14  13  14  15  16  15
16  15  16  15  14  13  12  11  12  11  10  11  12  11  12  13  14  13  14  15
 [157]  16  17  16  17  18  17  16  15  14  13  12  11  10  11  12  13  12  11  10
9   8   9   8   9  10   9   8   9  10   9   8   7   8   9  10   9   8   7   8
...
...
```

You can draw all this:

```
plot (x = steps.rw, type = 'l')
```

The figure is not here, because for each run the figure looks different. Argument x in the function `plot` contains values you want to draw. Theoretically, the arguments should be two, x and y, i.e. the values for x and y axis; if the y argument is missing, the function proceeds as following: the values of x argument are drawn on vertical axis, and horizontal axis is represented by the order of values in the vector.

Definition and the use of functions

Each time I run the random walk algorithm again, I got different result. How about to repeat the whole process many times, for example 25 times? We may simply nest the loop with the random-walk circle of 1000 steps into another loop repeating 25 times, but here we opt for another way. Let's create a function (let's call it `random.walk`), which will simply content the whole algorithm of random walk and which will be easily adjustable (i.e. I can modify number of steps to be done in each random walk procedure).

```
random.walk <- function (no.steps)
    {
    steps.rw <- vector (mode = 'numeric', length = no.steps)
```

```
    for (k in seq (2, no.steps))
      {
      steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
      }
    }
```

As you can see, the newly created function is called `random.walk`, and contains the only argument named `no.steps` - number of steps needed to be done. In the first step, the function creates an empty object `steps.rw`, which is a numeric vector (will be filled by numbers) of the length equal to the number of steps. Then the function runs the loop, starting from the second step (the first step is zerro, and it doesn't need to be defined, since newly created `steps.rw` contains all zerroes). The logic of the loop is identical as the one described above.

Let's try to use it:

```
random.walk (1000)
```

The function will run, but won't return any result. The reason is that `steps.rw` object was really created within the loop, but remained inside the function, since I didn't explicitly called it out. The way to fix it is that in the end of the function, I simply call the object, and in this way it will be sent out of the function (to the screen or to the object to which I assign the function).

```
random.walk <- function (no.steps)
    {
    steps.rw <- vector (mode = 'numeric', length = no.steps)
    for (k in seq (2, no.steps))
      {
      steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
      }
    steps.rw
    }
```

Even better solution is to *return* the value of the object `step.rw` by the function `return`; this is standard component of the function if that one should return some values, and whatever operation which is written after the `return` command in the body of the function is not evaluated:

```
random.walk <- function (no.steps)
    {
    steps.rw <- vector (mode = 'numeric', length = no.steps)
    for (k in seq (2, no.steps))
      {
      steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
      }
    return (steps.rw)
    }
```

Just a small sidenote - if I don't want to write the function all from the beginning, I can just use the original one and edit it, using the command `edit`:

```
random.walk <- edit (random.walk)
```

Edited version should be assigned to some object (could be the same as the name of the function) to make sure that changes will be saved (if I use `edit (random.walk)`, when closing the editing window I will be prompted whether I want to save the changes, but even if I did, they won't get saved to the original function which is being edited). Alternative is the function `fix`, which directly saves the changes into edited objects.

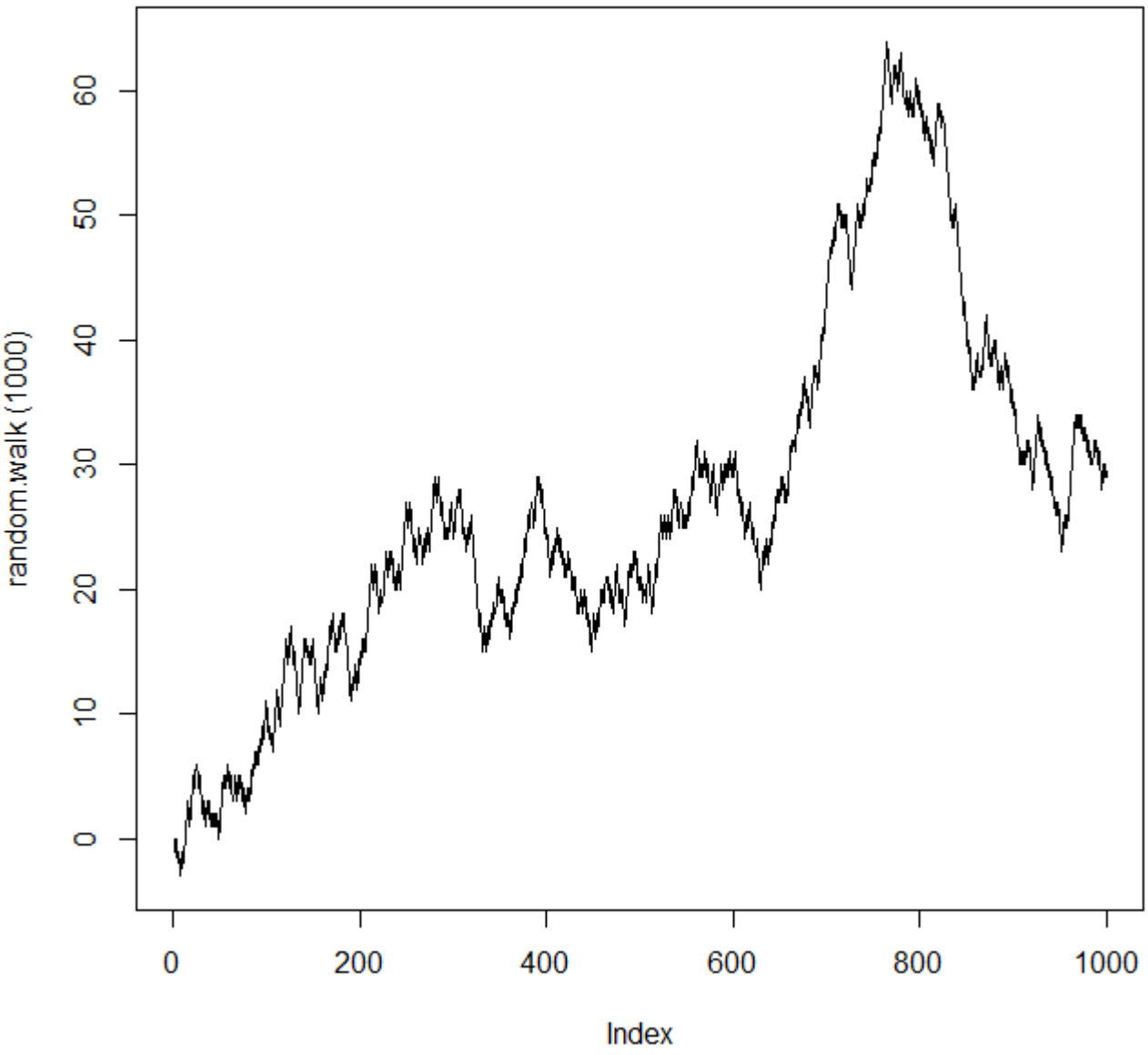
Let's see again how does the function works (I use smaller number of steps, so as the resulting vector is not too long):

```
random.walk (100)
```

```
  [1]   0  -1   0   1   2   1   0  -1  -2  -1   0  -1   0  -1  -2  -3  -4  -5  -4
-5  -4  -3  -4  -5  -6  -5  -6  -7  -8  -9 -10  -9  -8  -9  -8  -7  -8  -9 -10
 [40] -11 -12 -13 -14 -15 -14 -13 -12 -11 -12 -11 -12 -13 -14 -13 -12 -13 -12 -13
-12 -11 -12 -13 -12 -11 -10 -11 -12 -11 -10  -9  -8  -9  -8  -9 -10 -11 -12 -13
 [79] -12 -13 -14 -13 -14 -13 -12 -13 -14 -13 -14 -13 -12 -13 -14 -13 -14 -13 -14
-15 -16 -15
```

I can directly plot it (with 1000 steps):

```
plot (random.walk (1000), type = 'l')
```



Summary of the R code

```
random.walk <- function (no.steps)
    {
    steps.rw <- vector (mode = 'numeric', length = no.steps)
    for (k in seq (2, no.steps))
      {
      steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
      }
    return (steps.rw)
    }
plot (random.walk (1000), type = 'l')
```

Extras: random walk repeated 25 times (+ some graphical adjustements)

And now the figure where several random walks are drawn at once. First, I need to split the graphical window into 25 subpanels (five rows and five columns), which can be done by setting up the graphical parameter `mfrow` using the function `par`. Additionally, I want to reduce the margins of each figure in each subpanels to minimum (I don't need to draw any labels and tickmarks) - this can be done by setting the graphical parameter `mar`, again using `par` function:

```
par (mfrow = c (5,5))
par (mar = c (0,0,0,0))
```

Finally, create the loop which will draw all 25 figures:

```
for (i in seq (1, 25))
    {
    plot (random.walk (1000), type = 'l', ann = F, axes = F)
    box ()
    }
```

In this cycle in fact we don't need the object `i`, but we need to use it within the `for` loop to make sure that the cycle will repeat exactly 25 times. In the plotting function `plot` are two new arguments" `ann = F`, ensuring that the annotation of axes will not be drawn, and `axes = F`, ensuring that axes and tickmarks will not be drawn. In the end, we add function `box ()`, which draws the box around the plotting region.

Summary of the random walk script

```
set.seed (212121)  # set up this number (the random seed) the same - you will get
the same result as is here

random.walk <- function (no.steps)
{
  steps.rw <- vector (mode = 'numeric', length = no.steps)
  for (k in seq (2, no.steps))
  {
    steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
  }
```

```
  steps.rw
}


par (mfrow = c (5,5))
par (mar = c (0,0,0,0))

for (i in seq (1, 25))
{
  plot (random.walk (1000), type = 'l', ann = F, axes = F)
  box ()
}
```

And the resulting figure:



Script for 3D random walk

Something extra: How to create the same in 3D space? You will need library `rgl` for it.

```
library (rgl) # if the library is not installed, use also install.packages ("rgl")

no.steps <- 10000
steps.rw <- matrix (0, ncol = 3, nrow = no.steps)
for (k in seq (2, no.steps))
  {
  steps.rw[k,] <- steps.rw[k-1,]
  which.to.add <- sample (1:3, 1)
  steps.rw[k, which.to.add] <- steps.rw[k, which.to.add] + sample (c (-1,1), 1)
  }

rgl.linestrips (steps.rw)

# If you want to make the whole process like a movie, use the following cycle:
# for (i in seq (2, no.steps)) rgl.linestrips (steps.rw[1:i,])
```